Amendments to the Specification

Please replace the paragraph on Page 4, lines 10 - 18 with the following marked-up replacement paragraph:

-- The sample markup document 200 in Fig. 2 defines a valid person element 210 that conforms ~~conform~~ to this base schema 100. The syntax at 220 of this sample document identifies the schema to which the document conforms. That is, according to the W3C documents defining the schema notation, the value of the "schemaLocation" attribute shown at 220 is used to "provide hints" as to where the schema can be found. In this example, the schema is identified using a Uniform Resource Identifier ("URI") with "base.xsd" as the resource name, and might therefore refer to the sample schema 100 in Fig. 1. The manner in which the base schema 100 of Fig. 1 may be extended to support alternative syntax and structures in conforming structured documents will now be described. --

Please replace the paragraph on Page 9, lines 1 - 13 with the following marked-up replacement paragraph:

-- Furthermore, schema extensions may be cumulative (i.e., nested), which exacerbates this problem for prior art parsers. Suppose, for example, that the schema extension 330 in Fig. 3B referred to the location of the schema extension 300 in Fig. 3A as its base (e.g., by specifying an attribute such as "... schemaLocation= ".../ext1.xsd" at 340), and the schema extension 360 in Fig. 3C referred to the schema extension 330 as its base (e.g., by specifying "... schemaLocation= ".../ext2.xsd" at 370). In that case, a valid XML document could contain person elements having gender, age, and marital status attributes (in addition to the height and weight attributes from the

base schema definition 100). Fig. 6 illustrates, in a composite form, a schema 600 that corresponds to the result of applying these nested extensions. (Note that this schema document 600 is provided only for illustrative purposes. The schema extensions still remain in distinct documents, as in Figs. 3A - 3C.) A document conforming to this nested extended schema is illustrated at 700 in Fig. 7. Pictorially, the nested extensions and their cumulative or composite effect are illustrated in Fig. 8 (see, generally, reference number 800). --

Please replace the paragraph that begins on Page 19, line 9 and carries over to Page 20, line 1 with the following marked-up replacement paragraph:

-- Fig. 10 provides another example of this selective specification of abstraction levels. Here, the schema extensions have been defined as cumulative (as was described with reference to Fig. 8). In the example of Fig. 10, schema extension 1010 extends base schema 900; schema extension 910 then further extends the result; and schema extension 1020 then further extends that result. So, for example, a person element might be defined to include a gender attribute by schema extension 1010, an age attribute by schema extension 910, and to also include a marital status attribute by schema extension 1020. Accordingly, the parser can validate the full syntax of a source document such as document 700 of Fig. 7. Consumer 1 in the example of Fig. 10 requests to receive objects according to the base schema (where the only attributes for a person element are height and weight attributes; see reference number 1030), while Consumer 2 requests to receive objects according to the second level of extensions (i.e., person elements having gender and age attributes, in addition to height and weight attributes; see reference number 1031), and Consumer 3 requests to receive objects only according to the first level of extensions (i.e., person

elements having height, weight, and gender attributes; see reference number 1032).

--

Please replace the paragraph on Page 20, lines 6 - 17 with the following marked-up replacement paragraph:

-- Fig. 11 illustrates how preferred embodiments notify a parser [[of]] to render (i.e., provide to a requesting consumer application) objects created according to a particular schema extension level. See reference number 1100. Prior art parsers are typically implemented with an interface that allows invocation of a "setFeature" method, whereby parser features or options may be selected by an application that instantiates a parser instance. Two invocations of this well-known setFeature method are shown in Fig. 11, at 1120 and 1130. The first invocation 1120 is known in the prior art, and specifies a validation feature that engages schema validation in the parser. A URI is specified as a parameter to the setFeature method, providing a fully-qualified reference to the validation feature. (The syntax shown at 1110 instantiates a new parser, then registers a content handler to handle parsed events and an error handler to handle validation errors, using prior art techniques which do not form part of the present invention.) The invocation at 1130 engages techniques defined by the present invention, as will now be described in more detail. --

Please replace the paragraph on Page 23, lines 9 - 17 with the following marked-up replacement paragraph:

-- Commonly-assigned U. S. ~~Patent~~ _____ (~~serial~~ Patent Application number

10/403,342[[,]] (attorney docket RSW920030004US1, filed 3/28/ 2003; now abandoned), which is titled "Dynamic Data Migration for Structured Markup Language Schema Changes", defines techniques for dealing with schemas that are undergoing revision. Using techniques disclosed therein, the XML files that adhere to a changing schema can by revised programmatically, using knowledge of the particular schema changes that have been made. (This knowledge also enables determining whether any validation problems that arise are simply due to the schema changes, or instead signify an error in the document-producing logic.) However, this commonly-assigned invention does not disclose use of selectable abstraction levels as disclosed herein. --

Please replace the paragraph that begins on Page 23, line 18 and carries over to Page 24, line 11 with the following marked-up replacement paragraph:

-- Commonly-assigned U. S. ~~Patent _____ (serial~~ Patent Application number 10/016,933[[,]] (attorney docket RSW920010220US1), which is entitled "Generating Class Library to Represent Messages Described in a Structured Language Schema", discloses techniques whereby class libraries are programmatically generated from a schema. Templates are used for generating code of the class libraries. According to techniques disclosed therein, optional migration logic can be programmatically generated to handle compatibility issues between multiple versions of an XML schema from which class libraries are generated. Multiple versions of an XML schema are read and compared, and a report of their differences is prepared. The differences are preferably used to generate code that handles both the original schema and the changed version(s) of the schema. The class library is then preferably programmatically re-generated such that it includes code for the multiple schema versions. This allows run-time

functioning of code prepared according to any of the schema versions. The techniques disclosed therein are not directed toward casting objects at selectable levels. --